An Approximation algorithm for scheduling Trees of Malleable Tasks

Renaud Lepère, Grégory Mounié, Denis Trystram

An Approximation algorithm for scheduling Trees of Malleable Tasks

Renaud Lepère, Grégory Mounié and Denis Trystram

Laboratoire ID (Informatique et Distribution) - IMAG
51 rue J.Kuntzmann
38330 Montbonnot St.Martin, France
{Renaud.Lepere, Gregory.Mounie, Denis.Trystram}@imag.fr
contact author: Denis.Trystram@imag.fr

Abstract. This work presents an approximation algorithm for scheduling the tasks of a parallel application. These tasks are considered as *malleable tasks* (MT in short), which means that they can be executed on several processors. This model receives recently a lot of attention, due mainly to their practical use for implementing actual parallel applications. Most of the works developed within this model deal with independent MT for which good approximation algorithms have been designed. This work is devoted to the case where MT are linked by precedence relations. We present a 4 $(1+\epsilon)$ approximation algorithm (for any fixed ϵ) for the specific structure of a tree. This preliminary result should open the way for further investigations concerning arbitrary precedence graphs of MT.

keywords. parallel computing - scheduling - malleable tasks - precedence constraints - trees.

1 Introduction

Since the eighties, many works have been developed for parallelizing actual large scale applications. Parallel implementations are based on algorithmic studies where scheduling and load-balancing issues are central points to be considered. It should be noticed that there exists a very large literature addressing the problem of scheduling efficiently the tasks of a parallel program. It corresponds to find at what time and on which processor the tasks will be executed. Many works have been developed including theoretical studies on abstract and idealized models and practical tools consisting of actual implementations on most existing parallel and distributed platforms.

Among the various possible approaches, the most commonly used is to consider the tasks of the program at the finest level of granularity and apply some adequate clustering heuristics for reducing the relative communication overhead [5]. It is well-known that problems where communications are taken into account lead generally to harder algorithms than without communications: for large communication delays no constant approximation algorithm is known at this time.

It is of course even more crucial for computational models with a finer communication representation like LogP [3]. Due to the intractability of the problem, the impact of the parallelization overhead is usually ignored.

Recently, a new computational model called *Malleable tasks* (MT) has been proposed as a promising alternative to standard delay models. MT are computational units which may be themselves executed in parallel. Communications are taken into account implicitly by a penalty factor.

The main result of this paper is to propose an approximation algorithm for scheduling MT in the presence of precedence constraints. Our approach is based on a two phase algorithm consisting first in computing, for each task, the number of processors for its execution (allotment phase) then, scheduling the corresponding multiprocessor tasks (tasks with fixed number of processors). This approach applied to graphs structured as trees is one of the first result in the direction of arbitrary precedence task graphs. Trees are well-known structures where each node has at most one predecessor (or successor in the case of in-trees).

We first present briefly the model of MT and give some basic properties. Then, after a brief survey on related works, we present the algorithm and detail its analysis.

The motivation of the model was well emphasized in some previous works, so, we will not discuss here the advantages of MT. The reader may refer to [11] for more details. Informally, the problem of scheduling MT, is to find the number of processors alloted to each task and when to execute them. Figure 1 represents a task graph associated to one feasible scheduling. Then we discuss the existing works related to the problem of scheduling MT.

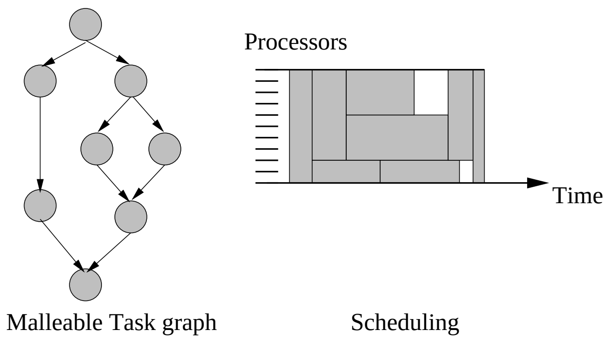

Fig. 1. A task graph and gantt chart of an associated scheduling.

Only few results are available for the scheduling MT problem, most of them concern independent tasks. Jansen and Porkolab prove that the problem with a fixed number of processors and independent MT admits a polynomial time approximation scheme [8]. It means that there exists a family of polynomial algorithms (A_ϵ) with a performance guarantee is $1+\epsilon$. However, this family is not a fully approximable scheme, and practically, the complexity makes it impossible to use.

In practice independent MT may be scheduled in two steps: computing first an *allotment* (which determines for all tasks the number of processors for their execution) and then, use some algorithms for scheduling multiprocessor tasks [15], for example with a 2-dimensional strip-packing algorithm [14].

There is very few works dealing with the problem of scheduling MT with precedence constraints. Prasanna et al. [13] proposed an algorithm for some specific structures of precedence task graphs, including trees for the continuous version of the problem (where an non-integer number of processors may be alloted to the MT). Let remark that they assume that the same speed-up function for all the tasks, which is a stronger restriction for the penalty functions than the monotonic ones described later. The speed-up is defined as the ratio of the execution time on one processor over the parallel execution time on any number of processors. Feldman et al. [4] study the effect of executing multiprocessor tasks on less processors than required. Their model can be seen as malleable tasks with a perfect or superlinear speed-up until a given threshold. In this context, constant approximation factor can be reached even for the case of an arbitrary task graph.

2 Problem definition

In this work, we consider that the parallel program is represented by a set of generic MT (computational units that may be themselves parallelized) linked by some precedence constraints, which are determined by the analysis of the data flow between the tasks. Determining this graph is usually done by the user, sometimes with the help of some software tools.

Let $G(V,E)$ be a directed graph where V represents the set of MT of cardinality n and E is the set of precedence constraints among the tasks. $t_{i,q}$ denotes the execution time of task $i \in V$ on q processors.

Definition 1 (Valid schedule). *A valid schedule* $\sigma = (date_\sigma, allot_\sigma)$ *is a pair of functions, from* V *to* $N \times [1,m]$*, respectively, that associated to each task* $i \in V$*, a date of execution* $date_\sigma \in N$ *(starting time), and a number of processors,* $allot_\sigma \in [1,m]$*, to execute it, such that at most* m *processors are engaged in the computation at a time, and all precedence constraints are respected:*

$$\forall i \in V, \qquad \sum_{j \in V, date_\sigma(i) \in [date_\sigma(j), date_\sigma(j)+t_{j,allot(j)})} allot_\sigma(j) \leq m$$

and

$$\forall (i,j) \in E : date_\sigma(j) \geq date_\sigma(i) + t_{i,allot_\sigma(i)}.$$

According to the standard notation, the *makespan* of schedule σ is denoted $\omega_\sigma = \max_{j \in V} date_\sigma(j) + t_{j,allot_\sigma(j)}$. As there is no ambiguity, we will forget the notation σ in all further expressions. In this paper, we consider the following problem:

Definition 2 (MT Scheduling problem (MTS)). *Find a valid schedule σ minimizing ω_σ.*

According to the behavior of a parallel program [2], people usually consider some hypotheses that simplify the analysis. They have been shown to be realistic while implementing actual parallel applications [1].

Monotonic penalty:

1. The execution time $t_{i,q}$ of a malleable task i is a monotonic decreasing function of the number of processors q executing the task: $t_{i,q+1} \leq t_{i,q}$
2. The work $w_{i,q} = q\ t_{i,q}$ of a malleable task i is a monotonic increasing function of q: $w_{i,q+1} \geq w_{i,q}$.

Practically, the first hypothesis means that adding some processors for executing a MT will decrease its execution time. This is realistic, at least until a threshold from which there is no more parallelism. It is easy to bound the number of processors from this moment. The second hypothesis reflects that the overhead for managing the parallelism usually increases with the number of processors.

Performance bounds:

For a particular allotment *allot*, two lower bounds of the makespan ω of any valid schedule σ may be easily computed

1. The total work $W_{allot} = \sum p_i t_i$ is the sum of the work of all tasks. For m processors, in any schedule σ, at least one processor compute at least W/m, namely the average work, thus $\omega \geq W_\sigma/m$,
2. The critical path length $C_{\max}$ is the maximum over all the paths P of G of $\sum_{i \in P} t_{i,allot(i)}$. As any valid schedule must respect the precedence constraints, $\omega \geq C_{max,\sigma}$.

The approximation algorithm that we propose in this paper is based on a two-phases approach. Its principle is given as follows:

Algorithm 1 Two phases MTS algorithm

```
Compute the allotment of all tasks
Schedule the obtained multiprocessor task graph
```

In the next section, we will first prove a preliminary result, which establishes that it is possible to schedule an arbitrary graph of multiprocessor tasks with a constant performance guarantee under small restrictions on the number of processors alloted to the tasks. Then, using this result, we will show that in the case of a tree, it is possible to construct an allotment (assign a number of processors to each task), such that both criteria of critical path and average work of the induced graph composed of multiprocessor tasks are bounded by the makespan of the optimal schedule.

3 The approximation result

3.1 Scheduling problem

We present an analysis for scheduling multiprocessor tasks. Let $G = (V, E)$ be the directed graph representing tasks together with their precedence constraints. Recall that a multiprocessor task requires a fixed number of processors for its execution. p_i and t_i denote respectively the number of processors and the execution time of task i using this number of processors. ω^∞ denotes the length of the critical path in graph G and W is the work of the tasks in V (equal to the sum of the works of the tasks $W = \sum p_i t_i$).

Let us now study the behavior of a list scheduling algorithm in regard to the maximum number of processors $\delta = \max_{i \in V} p_i$ needed to execute a multiprocessor task. This algorithm, whose principle is presented below, is a direct adaptation to the well-known list algorithm of Graham for sequential tasks [6]. Its objective is to compute for each task i its execution date $date(i)$. *Ready* denotes the set of tasks whose all predecessors have been scheduled.

The earliest execution $date(i)$ is the smallest date such that all the predecessors of i have already been executed (and thus i is in *Ready*) and at least p_i processors are available.

Algorithm 2 List scheduling for multiprocessor tasks.

```
Compute the set of ready tasks Ready
while Ready ≠ ∅ do
  for all i ∈ Ready do
    Compute the earliest date of execution date(i) of task i.
  end for
  Schedule any of the tasks in Ready with the minimal date(i).
  Update Ready.
end while
```

Proposition 1. *The makespan ω_L of a schedule obtained by algorithm 2 is bounded by:* $\frac{W+(m-\delta)\omega^\infty}{m-\delta+1}$.

Proof: The result is a generalization of the well-known bound of Graham [6]. The interval $[0, \omega_L]$ is partitioned into two sets of time slots, namely, I^+ and I^-. They correspond respectively to the times where strictly more than $m - \delta$

processors are occupied and at least δ processors are not occupied. Note that at least one processor is occupied at any time. In the following, $|I|$ will denote the duration of (non-continuous) period I.

The total work performed during periods I^+ and I^- are greater than $(m - \delta + 1)|I^+|$ and $|I^-|$, respectively. Thus,

$$W \geq (m - \delta + 1)|I^+| + |I^-|.$$

To obtain the result, the idea is to bound the term $|I^-|$. Let us prove that $|I^-| \leq \omega^\infty$. Intuitively there are at each time at least δ free processors and thus there is no conflict between processors in I^-.

More formally, we build a path in the transitive closure of graph G. Starting from the end of the schedule, we consider a multiprocessor task x_1 that finishes its execution at time ω_L and build recursively a task sequence in G.

This is illustrated on figure 2. There exists a precedence relation between x_2 and x_1, otherwise x_1 could have been started at the beginning of the second I^- slot. For the construction of x_3 from x_2, the dark grey tasks represent independent tasks that are executed during the date τ_2. They are candidates to be predecessors of x_1 and x_2, otherwise x_2 could have been started in the first I^- slot.

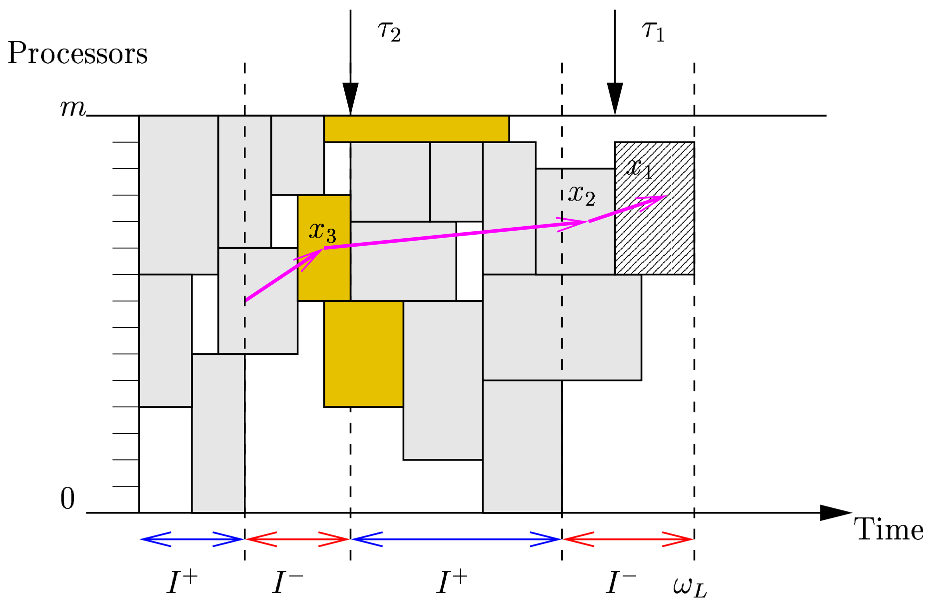

Fig. 2. A multiprocessor task schedule obtained by a list algorithm and the construction of a partial task sequence

Suppose that the partial task sequence $(x_i, x_{i-1}, ..., x_1)$ is built and $I^- \backslash [date(x_i), \omega_L]$ is not empty, we add x_{i+1} as one of the tasks that are executed at time $\tau_i = \max(I^-$
$\backslash [date(x_i), \omega_L])$. There exists at least one such task which is a predecessor of x_i (otherwise, task x_i should have been scheduled before by the list algorithm). We repeat the process until $I^- \backslash [date(x_i), \omega_L]$ becomes empty. This defines a task sequence $C = (x_k, \cdots, x_1)$, and then an oriented path of G, such that at each $t \in I^-$ there is a task of C that is being executed. Thus, we have:

$$|I^-| \le \omega^\infty.$$

By reporting this value into the previous expression, we obtain easily:

$$\omega_L = |I^+| + |I^-| \le \frac{W + (m-\delta)\omega^\infty}{m - \delta + 1}$$

Corollary 1. *The performance guarantee of any list algorithm for scheduling multiprocessor tasks is:* $\frac{(2m-\delta)}{m-\delta+1}$ *where* δ *is the maximum number of processors used for executing the tasks.*

Let us remark that this guarantee corresponds to the Graham's bound $2 - 1/m$ for sequential tasks where $\delta = 1$. This bound is large when $\delta = m$, however, if all the tasks use only half of the available processors (that is when $\delta = m/2$), the guarantee is bounded by 3.

3.2 Allotment problem

In this section, we focus on the problem of finding a good allotment. The main idea is to determine an allotment such that the maximum between both opposite criteria of length of critical path and average work is minimized. First, we will present more formally the problem of the allotment, and we will show how to derive an approximation algorithm for the MTS problem. We will show later that for trees it is possible to find a fully polynomial approximation scheme.

Let us consider an allotment function *allot*. In the following, the length of critical path of the corresponding multiprocessor task graph is denoted ω^∞_{allot} and it is defined as the maximum over all the paths P of the oriented weighted G of $\sum_{i \in P} t_{i,allot(i)}$. This notation is used because the length of the critical path is also the makespan of the schedule on an infinite number of processors. $W_{allot} = \sum_{i \in V} allot(i) t_{i,allot(i)}$ denotes the total work. Now let consider the following problem:

Definition 3 (allotment problem). *Find an allotment function* $allot : V \to [1, m]$ *such that* $max(\omega^\infty_{allot}, W_{allot}/m)$ *is minimized.*

The following property holds.

Property 1. The optimal value of the allotment problem is smaller than the optimal makespan ω^* of the MTS problem: $max(\omega^\infty_{allot^*}, W_{allot^*}/m) \le \omega^*$ where $allot^*$ is the optimal solution of the allotment problem

The proof is straightforward: for any allotment function *Allot* of an optimal schedule including the optimal solution of the allotment problem, we have, by definitions, $max(\omega^{\infty}_{allot}, W_{allot}/m) \leq \omega^*$.

We will now show how to obtain an approximation algorithm for MTS from an approximation algorithm for the allotment problem.

Proposition 2. *Given a k-approximation algorithm for the allotment problem, we can construct a $4k$-approximation algorithm for the MTS problem.*

Proof. The main idea is to use the result of proposition 1 that gives an approximation for the multiprocessor scheduling problem. Let *allot* be the allotment found by the k-approximation algorithm. We construct *allot'* such that all tasks in *allot'* never use more than δ processors, where δ is a given number in $[1, m]$. More formally, for any task i, $allot'(i) = allot(i)$ if $allot(i) < \delta$ and $allot'(i) = \delta$ otherwise. The makespan provided by Algorithm 2 for scheduling G with the fixed allotment *allot'* is according to Proposition 1 bounded by:

$$\omega_{allot'} \leq \frac{W_{allot'} + (m - \delta)\omega^{\infty}_{allot'}}{m - \delta + 1}.$$

Using the monotonic assumption described in section 2, we get $\omega^{\infty}_{allot'} \leq \frac{m}{\delta}\omega^{\infty}_{allot}$ and $W_{allot'} \leq W_{allot}$. Since according to Property 1 we have $\omega^{\infty}_{allot} \leq k\omega^*$ and $W_{allot}/m \leq k\omega^*$, we finally obtain:

$$\frac{\omega_{allot'}}{\omega^*} \leq \frac{km^2}{\delta(m - \delta + 1)}.$$

This ratio is minimal for $\delta = m/2 + 1$, and in this case we have: $\frac{\omega_{allot'}}{\omega^*} \leq \frac{4k}{(1+1/m)^2} \leq 4k$.

4 Application to trees

Finally, it remains to determine how to find an approximation scheme for solving the allotment problem for the case of trees. For this purpose, we will use a k-dual approximation method introduced by Shmoys et Hochbaum [7]. Let us give an integer λ, we consider the following problem:

Definition 4 (Constrained allotment problem). *Find an allotment function allot such that the length of the critical path ω^{∞}_{allot} is smaller than λ and the average work W_{allot}/m is as small as possible.*

In the next section, we will show that this problem can be solved optimally using dynamic programming for the case of trees, and deduce a polynomial-time algorithm that computes an allotment function $allot_{\epsilon}$, for a given $\epsilon > 0$, whose work is smaller than the work of an optimal allotment and whose critical path length do not exceed $\lambda(1 + \epsilon)$. This can be seen as an optimal solution of the same problem using more resources: a widely used technique in on-line algorithm [9, 12]. We call this algorithm a $(1+\epsilon)$-optimal algorithm. Firstly, we will show that if we can find a k-optimal algorithm for the constrained allotment problem, then we can find a $k(1+\epsilon)$-approximation for the allotment problem.

Proposition 3. *Given a k-optimal algorithm for solving the constrained allotment problem, we can construct a $k(1+\epsilon)$-approximation algorithm for the allotment problem.*

Proof: Consider an optimal solution of the allotment problem denoted by $allot^*$. Let η^* be the maximum between the length of critical path and the average work for this allocation: $\eta^* = \max(\omega^{\infty}_{allot^*}, W_{allot^*}/m)$. Let consider the monotonic behavior of the penalty, we have: $\omega^1/m \leq \eta^* \leq \omega^1$, where ω^1 is the execution time of the graph on a single processor $(\sum_{i \in V} t_{i,1})$. Given a real number $\epsilon > 0$, consider a partition of the interval $[\omega^1/m, \omega^1]$ into $\lceil (m-1)/\epsilon \rceil$ intervals of size $\frac{\omega^1.\epsilon}{m}$. Considering that we solve the constrained allotment problem with k-optimal algorithm, at the beginning of one of the interval λ. Let $allot$ be the solution found by this algorithm, and $\overline{allot}^*$ be the optimal solution. By definition, we now have $\omega^{\infty}_{allot} \leq k\lambda$ and $W_{allot}/m \leq W_{\overline{allot}^*}/m$. If $W_{allot}/m \leq \lambda$, then obviously $\eta^* \leq k\lambda$. If $W_{allot}/m > \lambda$, then $W_{\overline{allot}^*}/m > \lambda$ thus $\eta^* > \lambda$. Finally, using a dichotomic search on λ, we can find in $\log_2((m-1)/\epsilon + 1)$ steps an interval $[\eta, \eta + \frac{\omega^1\epsilon}{m}]$ and an allotment function $allot$ (which corresponds to the allotment function found by the k-optimal algorithm choosing $\lambda = \eta + \frac{\omega^1\epsilon}{m}$) such that $\eta \leq \eta^* \leq k(\eta + \frac{\omega^1\epsilon}{m})$. $\omega^{\infty}_{allot} \leq \eta + \frac{\omega^1\epsilon}{m}$, and $W_{allot}/m \leq k(\eta + \frac{\omega^1\epsilon}{m})$, thus $\omega^{\infty}_{allot}/\eta^* \leq 1+\epsilon$ and $W_{allot}/m/\eta^* \leq k(1+\epsilon)$. Thus, $allot$ is a $k(1+\epsilon)$-approximation for the allotment problem.

4.1 Dynamic Programming

In this section, we detail how the constrained allotment problem can be solved using a dynamic programming algorithm for the case of trees. Let us consider an in-tree $G = (V, E)$ with n nodes labeled according to any partial order of the precedences. We denote by $prec(i)$ the set of immediate predecessors of task i. $W(i,t)$ is computed as the minimum work required for executing the i first tasks with a length of critical path lower than t. $W(i,t)$ is set to $+\infty$ if it is not possible to execute the i first tasks with a critical path lower than t. The algorithm is given in figure 4.1. The tasks are sorted in respect to the precedence constraints.

The solution of this problem is $W(n, \lambda)$. It is computed in time $nm\lambda$. However, as λ is potentially very large, the result is not fully satisfactory.

4.2 Approximation for the constrained allotment problem

Now, we are looking for a $(1+\epsilon)$-optimal algorithm for the constrained allotment problem. Let us consider the similar problem after rounding the execution time:

Algorithm 3 Dynamic Programming algorithm for the constrained allotment problem for trees

$W(0,t) = 0$ for all $t \geq 0$ and $+\infty$ otherwise.
for all $t \in [1, \lambda]$ **do**
  for all $i \in [1, n]$ **do**
    $W(i,t) = min_{q=1..m} w_{i,q} + \sum_{j \in prec(i)} W(j, t - t_{i,q})$
  end for
end for

$\widehat{t}_{i,q} = \lfloor t_{i,q}/c \rfloor$ and $\widehat{\lambda} = \lambda/c$ where c is a constant greater than 1 that will be fixed later. Consider that $\widehat{allot}$ is the optimal solution (obtained by the dynamic programming algorithm) of this rounded problem and $\overline{allot}^*$ the optimal solution of the original problem. It is obvious that $W_{\widehat{allot}} \leq W_{\overline{allot}^*}$. We denote by $Paths$ the set of all the paths of the graph G. We bound now the length of critical path of $\widehat{allot}$ by bounding the quantity of work neglected by the rounding.

$$\omega^{\infty}_{\widehat{allot}} = \max_{P \in Paths} \sum_{i \in P} t_{i,\widehat{allot}(i)} \leq n\ c + c \max_{P \in Paths} \sum_{i \in P} \widehat{t}_{i,\widehat{allot}(i)} \leq n\ c + \omega^{\infty}_{\overline{allot}^*}$$

Thus, we get $\omega^{\infty}_{\widehat{allot}} \leq n\ c + \lambda$. If we choose $c = \frac{\epsilon\,\lambda}{n}$, we obtain: $\omega^{\infty}_{\widehat{allot}} \leq \lambda(1+\epsilon)$. Thus such a rounding process leads to a $(1+\epsilon)$-optimal algorithm for the constrained allotment problem whose computation time is in $\mathcal{O}(\frac{m\,n^2}{\epsilon})$.

On one hand, considering Proposition 3, we can conclude that for the special case of trees the allotment problem admits a fully polynomial time approximation scheme whose running time is in $\mathcal{O}(\log(\frac{m}{\epsilon})\frac{m^2\,n}{\epsilon})$.

On the other hand, from Proposition 2, if we know how to solve the allotment problem with a guarantee of k, then we know that it is possible to solve MTS with a guarantee of $4k$. Finally, we deduce that there exists a solution of the MTS problem with a guarantee $4(1+\epsilon)$.

5 Conclusion

In this paper, we have studied the problem of scheduling malleable tasks in the presence of precedence constraints. We designed an algorithm with a guarantee $4\,(1+\epsilon)$ in time $\mathcal{O}\left(\log\left(\frac{m}{\epsilon}\right)\ \frac{n^2\,m}{\epsilon}\right)$ for trees.

This algorithm could be used for implementing some actual problems whose computations are organized as trees like Sparse Cholesky Factorization [10]. The approximation ratio was established for the worst case and we expect a much better average behaviour. These preliminary theoretical results could open the way for the designers to obtain good approximation algorithms for arbitrary precedence graphs of malleable tasks.

References

1. E. Blayo, L. Debreu, G. Mounié, and D. Trystram. Dynamic load balancing for ocean circulation model with adaptive meshing. In Patrick et al. Amestoy, editor, *Euro-Par' 99 Parallel Processing - 5th International Euro-Par Conference*, number 1685 in Lecture Notes in Computer Science, pages 303–312, August 31 - September 3 1999.
2. M. Cosnard and D. Trystram. *Parallel algorithms and architectures.* International Thomson Publishing, 1995.
3. D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken. Logp: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, 1996.
4. A. Feldmann, M-Y. Kao, and J. Sgall. Optimal online scheduling of parallel jobs with dependencies. In *25th Annual ACM Symposium on Theory of Computing*, pages 642–651, San Diego, California, 1993. url: http://www.ncstrl.org, CS-92-189.
5. A. Gerasoulis and T. Yang. PYRROS: static scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th ACM International Conference on Supercomputing*, pages 428–437. ACM, jul 1992.
6. R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.
7. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.
8. K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (SODA 98)*, pages 490 – 498, San Francisco, CA USA, January 25 - 27 1998.
9. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science*, pages 214–221, Milwaukee, Wisconsin, 23–25 1995. IEEE.
10. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Cummings, 1994.
11. R. Lepère, G. Mounié, D. Trystram, and B. Robič. Malleable tasks: An efficient model for solving actual parallel applications. In E. H. D'Hollander, G. R. Joubert, F. J. Peters, and H. J. Sips, editors, *Parallel Computing: Fundamentals and Applications. Proceeding of the International Conference Parco'99*, pages 598–605, Delft, The Netherlands, August 1999. Imperial College Press.
12. C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 1997.
13. G. N. Srinivasa Prasanna and B. R. Musicus. Generalised multiprocessor scheduling using optimal control. In *3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 216–228. ACM, 1991.
14. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.
15. J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332, 1992.